BIS L-870 handheld driver for Windows CE 5.0
User's Manual

Table of content

1 Introduction 3
2 Additional files to the project 4
3 Header files 4
3.1 BISReader.h 4
4 Driver functions in details 4
4.1 BIS_ComPortState 5
4.2 BIS_OpenCOMPort 5
4.3 BIS_OpenReader 5
4.4 BIS_CloseReader 5
4.5 BIS_CloseCOMPort 6
4.6 BIS_InitReader 6
4.7 BIS_ReadData 6
4.8 BIS_ReadTagUID 7
4.9 BIS_WriteData 7
4.10 BIS_WritePattern 7
4.11 BIS_InitDataCarrier 8
4.12 BIS_ResetReader 8
4.13 BIS_LOCKCodeTag 8
4.14 BIS_AntennaPowerOn 8
4.15 BIS_AntennaPowerOff 9
4.16 BIS_GetDllVersion 9
4.17 BIS_GetLastError 9
4.18 BIS_GetLastErrorText 9
4.19 BIS_UnLoadUSB 10
4.20 BIS_ReLoadUSB 10
4.21 BIS_PowerOffRWHead 10
5 Usage of the driver (examples) 10
5.1 Preparation 10
5.2 Open a BIS module on port 11
5.3 Read from the code tag 11
5.4 Write to the code tag 11
5.5 Read UID of the code tag 11
5.6 Lock code tag 11

1 Introduction

BIS_L_WinCE_DLL.dll is a driver, written for BIS L-870, which covers the user's command set of *Balluff Dialog* protocol and also provides some extra functionality to the user under *Windows CE* environment.
The driver is written in native C++ code and designed for Windows CE 5.0. The functions are exported in C format, ignoring name-mangling.
It is prepared for use of 2 virtual communication ports (VCP), which has the driver from *Silabs*. (https://www.silabs.com)

The Read/Write head module connects to the handheld via USB port, thus the functions of the USB Host (like *open, close, read, write* etc.) is reached through function-calls of the provided USB driver interface (SIUSBXP_LIB.dll).

Implemented command set of Balluff Dialog:

- *ReadData* – read data from data carrier
- *WriteData* – write data to data carrier
- *WritePattern* – write a constant character to the data carrier
- *ReadTagID* – read UID of the data carrier
- *InitDataCarrier* – init data carrier for CRC16
- *InitReader* – set configuration of the processor
- *ResetReader* – reset read/write processor
- *LOCKCodeTag* – locks the code tag, changing it to read-only
- *AntennaPowerOn* – turns on power of the antenna of the read/write head
- *AntennaPowerOff* – turns off power of the antenna of the read/write head

There are other functions which are not part of the Balluff Dialog protocol, but can be called via the driver:

- *ComPortState* – gets the state of the COM port (open, closed, exist etc.)
- *OpenCOMPort* – opens the port
- *CloseCOMPort* - closes the port
- *OpenReader* – initializes and tests the read/write head
- *CloseReader* – closes the read/write head
- *GetDLLVersion* – gets the version info of the driver (version number, release date, BIS system type)
- *GetLastError* – gets the error number of the last error within the driver
- *PowerOffRWHead* – turns off the power of the USB port, thus the RW head
- *UnloadUSB* – close the USB port, but keeps the connection handle to opened BIS
- *ReloadUSB* – re-opens the USB port to use the original connection handle

2 Additional files to the project

As it was mentioned before, the driver uses USB driver from Silabs. Thus the USB driver interface has to be added to the project:

- *SIUSBXP_LIB.dll* USB driver interface

This driver is written specially for pocket PC from *PSION Teklogix*. When user turns on the PC, the USB port is *not powered*, thus the driver has to power on the port. In order to user can access the API collection of PSION WorkaboutPro he has to add the following file to the project as well:

- *PtxSdkCommon.dll* this is a library, containing API functions of PSION WAP

In order to user can use the functions of BIS L-870 reader, he has to add the driver and the appropriate headers to the project:

- *BIS_L_WinCE_DLL.dll* BIS L-870 driver for Windows CE 5.0
- *BIS_L_WinCE_DLL.lib* Static library to link the functions in the driver
- *BISReader.h* Contains function, type and return value declarations to BIS L-870 reader

At runtime user needs only 4 files: SIUSBXP_LIB.dll, which is an API collection to SIUSBXP.dll driver. (SIUSBXP.dll has to be installed in Windows folder), PtxSdkCommon.dll and BIS_L_WinCE_DLL.dll.
SIUSBXP.reg is the registration file to write the Registry. This file will create the entries for the USB driver.

3 Header files

This chapter contains detailed information about the header and source files in the project.

3.1 BISReader.h

This header contains the function declarations to use BIS L-870 reader and all the BIS L-870 related types, structures, enumerations and macros (definitions). All Balluff Dialog error codes and driver-related error codes are defined here.
User also has to add BIS_L_WinCE_DLL.lib static library to the project. This library file links the declared functions to the driver.

4 Driver functions in details

The user communicates with the BIS module via the function calls of the driver. The following chapter describes these functions.
In order to the user can use the drivers functionalities, he has to open a port with `BIS_OpenCOMPort` and open the BIS module with `BIS_OpenReader`. When functions return without error, user has a valid connection handler with which he can access the functions.
User should never change this connection handler manually, because he can't call the other functions after that.

The following section contains the list of the exported functions.

4.1 BIS_ComPortState

Checks the status of the given port.

Prototype: `int BIS_ComPortState(UINT PortNumber)`

Parameters: 1. *PortNumber* – the number of the USB port (virtual COM port)

Return value: `ERR_COMPORT_NOT_EXIST`
`ERR_COMPORT_USED`
`ERR_COMPORT_CLOSED`
`ERR_UNKNOWN`

4.2 BIS_OpenCOMPort

Opens the given port and gives back a Connection Handle.

Prototype: `int BIS_OpenCOMPort(UINT PortNumber, CONNHND& ConnHnd)`

Parameters: 1. *PortNumber* – the number of the USB port
2. *ConnHnd* – pointer to a connection handle parameter

Return value: `ERR_COMPORT_CLOSED`
`ERR_COMPORT_OPENED`

4.3 BIS_OpenReader

Opens the BIS module on the previously opened port and sets the Connection Handle.

Prototype: `int BIS_OpenReader(CONNHND& ConnHnd, bool bCRC16, UINT uiDataCarrierType)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter
2. *bCRC16* – true if the BIS module is set to use CRC16 check
3. *uiDataCarrierType* – defines which data carrier types can be used for the reader

Return value: `ERR_READER_OPENED`
`ERR_READER_CLOSED`

4.4 BIS_CloseReader

Closes the opened BIS module, accordingly the Connection Handle.

Prototype: `int BIS_CloseReader(CONNHND& ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable

Return value: `ERR_READER_CLOSED`
`ERR_READER_OPENED`

4.5 BIS_CloseCOMPort

Closes the opened port, accordingly the Connection Handle.

Prototype: `int BIS_CloseCOMPort(CONNHND& ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter

Return value: `ERR_COMPORT_CLOSED`
`ERR_COMPORT_OPENED`

4.6 BIS_InitReader

Sets some parameters of the given opened BIS module, accordingly the Connection Handle.

Prototype: `int BIS_InitReader(CONNHND& ConnHnd, bool bCRC16, UINT uiDataCarrierType)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter
2. *bCRC16* – set the module to use CRC16 check
3. *uiDataCarrierType* – set the module to use the chosen data carrier types

Return value: `ERR_NOERROR`
`ERR_READER_INIT_FAILED`

4.7 BIS_ReadData

Reads data from the data carrier, starting from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_ReadData(CONNHND ConnHnd, UINT uiStartAddr, UINT Datalength, TCHAR* szDataBuffer)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter
2. *uiStartAddress* – start address of the data carrier to read from
3. *Datalength* – number of bytes to read (max. size of the szDataBuffer)
4. *szDataBuffer* – pointer to the data buffer where the data arrives back; Unicode characters

Return value: `ERR_NOERROR`
`ERR_READ`

4.8 BIS_ReadTagUID

Reads the ID and type of the data carrier; accordingly the Connection Handle. It also indicates if the data carrier is not presented.

Prototype: `int BIS_ReadTagID(CONNHND ConnHnd, DATACARRIERINFO& DCInfo )`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter
2. *DCInfo* – reference to the structure where the data carrier information will be stored

Return value: `ERR_NOERROR`
`ERR_READ`

4.9 BIS_WriteData

Writes data to the data carrier, starting from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_WriteData(CONNHND ConnHnd, UINT uiStartAddr, TCHAR* DataToWrite, UINT Datalength)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter
2. *uiStartAddress* – start address of the data carrier to write from
3. *DataToWrite* – user data to write onto the data carrier; Unicode characters
4. *Datalength* – the number of the desired bytes to write (max. size of DataToWrite)

Return value: `ERR_NOERROR`
`ERR_WRITE`

4.10 BIS_WritePattern

Writes a constant character to the data carrier, starting from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_WritePattern(CONNHND ConnHnd, UINT uiStartAddr, UINT Patternlength, TCHAR* Pattern)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter
2. *uiStartAddress* – start address of the data carrier to write from
3. Patternlength – the number of the bytes to write
4. *Pattern* – pointer to a char array which contains the pattern; Unicode

Return value: `ERR_NOERROR`
`ERR_WRITE`

4.11 BIS_InitDataCarrier

Initializes a data carrier with all 0 hex values, preparing it for use of CRC16. Starts from the given address in the given length; accordingly the Connection Handle.

Prototype: `int BIS_InitDataCarrier(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter

Return value: `ERR_NOERROR`
`ERR_DATACARRIER_INIT_FAILED`

4.12 BIS_ResetReader

Resets the given BIS module, accordingly the Connection Handle.

Prototype: `int BIS_ResetReader(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable

Return value: `ERR_NOERROR`
`ERR_UNKNOWN`
`ERR_HEAD_COMMUNICATION`
`ERR_NO_ANSWER`
`ERR_READ`

4.13 BIS_LOCKCodeTag

Locks the code tag, changing it to read-only. User has to give the new UID of the code tag. If the user wants to lock the tag with CRC check, he has to give only 3 byte-long UID, otherwise 5 bytes.

Prototype: `int BIS_LOCKCodeTag(CONNHND ConnHnd, bool WithCRC, BYTE* newTagUID)`

Parameters: 1. *ConnHnd* – pointer to a connection handle variable
2. *WithCRC* – write the new code tag UID with CRC
3. *newTagUID* – pointer to the new tag UID (3 or 5 bytes)

Return value: `ERR_NOERROR`
`ERR_LOCK_CODETAG`

4.14 BIS_AntennaPowerOn

Turns on the power of the given BIS module, accordingly the Connection Handle.

Prototype: `int BIS_AntennaPowerOn(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter

Return value: `ERR_NOERROR`
`ERR_TURN_ON_ANT`

4.15 BIS_AntennaPowerOff

Turns off the power of the given BIS module, accordingly the Connection Handle. When user turns off the antenna, the power consumption drops down dramatically, thus saves energy.

Prototype: `int BIS_AntennaPowerOff(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – pointer to a connection handle parameter

Return value: `ERR_NOERROR`
`ERR_TURN_OFF_ANT`

4.16 BIS_GetDllVersion

Retrieves information about the driver, such as date of release, version, BIS system type.

Prototype: `int BIS_GetDLLVersion(BISVERSION& Version)`

Parameters: 1. *Version* – pointer to the structure where the information will be stored

Return value: `ERR_DLL_VERSION_EXIST` (not an error)

4.17 BIS_GetLastError

Gives back the last error number after a function fails.

Prototype: `int BIS_GetLastError()`

Parameters: -

Return value: `The appropriate error number.`

4.18 BIS_GetLastErrorText

Gives back the last error as a character array after a function fails.

Prototype: `TCHAR* BIS_GetLastErrorText()`

Parameters: -

Return value: `The appropriate error with text.`

4.19 BIS_UnLoadUSB

Closes the USB port, but keeps the Connection Handle for later use. This function has to be called before the Pocket PC goes to Suspended mode, because it powers off the USB port. If the port stays opened, Windows stores its handle, and after the user turns on the PSION, he can't access the previously opened port. During a USB device enumeration user can find 2 devices, although only 1 exists in the Windows system.

Prototype: `int BIS_UnLoadUSB(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – the connection handle parameter

Return value: `ERR_NOERROR`
`ERR_UNLOAD_USB`

4.20 BIS_ReLoadUSB

Re-opens the USB port accordingly the Connection Handle. This function has to be called after the Pocket PC comes back from Suspended mode. This function re-opens the previously opened USB port, thus user can communicate with the same BIS L-870 module, without going through again the opening procedure.

Prototype: `int BIS_ReLoadUSB(CONNHND ConnHnd)`

Parameters: 1. *ConnHnd* – the connection handle parameter

Return value: `ERR_NOERROR`
`ERR_RELOAD_USB`

4.21 BIS_PowerOffRWHead

This functions turns off the power on USB port, thus powers off the BIS L-870 module.

Prototype: `int BIS_PowerOffRWHead()`

Parameters: -

Return value: `ERR_NOERROR`

5 Usage of the driver (examples)

In the following chapter the reader can find some code snippets, how to use the driver properly.

5.1 Preparation

Include the main header (BISReader.h), which contains the driver related definitions and the driver function declarations.

```
#include "BISReader.h"
```

Before user can use the driver, he has to declare a variable type of CONNHND. This is a connection handler, which contains information about the opened device on the port.

```
CONNHND MyConnection;
```

5.2 Open a BIS module on port

Before user can use a BIS module, he has to open the port and the given device. Open the device on port 1, as the USB port can be found on this.

```
BIS_OpenCOMPort(1, MyConnection);

BIS_OpenReader(MyConnection, CRC16_NO, BIS_CT_AllTypes);
```

5.3 Read from the code tag

Create a *DataBuffer*, type of TCHAR to store the retrieved data, and then call the appropriate function.

```
TCHAR DataBuffer[192];

BIS_ReadData(MyConnection, 0, 192, ReceivedData);
```

For parameter details, see chapter 4.

5.4 Write to the code tag

```
BIS_WriteData(MyConnection, 0, L"Text to write", 13);
```

The data has to be Unicode character string! The last parameter indicates how many characters are in the buffer; less can be given, but more may cause runtime errors!

5.5 Read UID of the code tag

Define and initialize a *DCInfo* buffer, type of DATACARRIERINFO to store the tag information, and then call the appropriate function.

```
DATACARRIERINFO DCInfo = {0};

BIS_ReadTagID(MyConnection, DCInfo)
```

For parameter details, see chapter 4.

5.6 Lock code tag

Call the appropriate function with the new UID of the code tag. If the user wants to generate CRC check for the new UID, give 3 bytes and set the second parameter to true, otherwise 5 bytes and the second parameter to false.

```
BYTE newUID[5] = {0x12, 0x37, 0x45, 0x91, 0x57};
```

```
BIS_LOCKCodeTag(MyConnection, CRC16_NO, newUID);
```

If the first 2 bytes of the new UID is not a valid CRC checksum and next time the reader is opened with CRC check, every function call will fail with CRC error which tries to access the code tag.